

Monitoraggio dinamico di entità geo-localizzate: un'applicazione web modulare

Michele Donati

`michele.donati5@studio.unibo.it`

Gennaio 2021

Il progetto verte alla realizzazione di un'applicazione web in grado di fornire uno strumento per il monitoraggio dinamico di entità geo-localizzate, sfruttando tecnologie come Express, Vue, Node.js e servizi esterni. L'obiettivo principale di questa applicazione è di permettere ad un suo utilizzatore di poter scegliere la tipologia di entità che desidera monitorare (e.g. veicoli, pazienti di un ospedale), configurando, tramite un file apposito, i criteri e i parametri per i quali queste entità vengono monitorate e/o evidenziate a schermo. La scelta di questi elementi tramite il file di configurazione permette anche, in maniera automatica e dinamica, la costruzione di un'interfaccia utente perfettamente e strettamente adattata alle esigenze espresse dall'utilizzatore. L'applicazione permette anche una continua contestualizzazione geografica tramite un processo di *reverse-geocoding*.

1 Obiettivi/Requisiti

L'obiettivo primario di questo progetto è quello di realizzare uno strumento grazie al quale un utente medio possa riuscire, nel modo più semplice e dinamico possibile, a monitorare ed analizzare il comportamento e/o le caratteristiche di entità geo-localizzate su una mappa.

Di seguito si riportano i requisiti:

- Il punto cardine dell'applicazione, allo stato attuale, deve essere un file di configurazione grazie al quale è possibile indicare: la tipologia di *entità* che si desidera monitorare, gli *indici* e i *criteri* con i quali monitorare e/o evidenziare le entità, le *caratteristiche* intrinseche delle entità che si vogliono visualizzare nell'interfaccia utente (e.g. per un veicolo possono essere: modello, tipo, casa manifatturiera, ergonomia, gasolio, ecc.).
- L'applicazione deve presentare un'interfaccia utente semplice e intuitiva, contenente una mappa per la visualizzazione in tempo reale delle entità tramite markers,

un tabella contenente le loro informazioni e degli indici di evidenziazione selezionabili.

- L'applicazione deve essere un'applicazione web modulare realizzata con le più recenti tecnologie, in modo da garantire, partendo dal file di configurazione, una perfetta dinamicità per la costruzione dell'interfaccia utente.
- Non avendo a disposizione dati reali, è necessario simularli.
- L'applicazione deve poter garantire un servizio di *reverse-geocoding*, quindi poter tradurre in tempo reale le coordinate geografiche delle entità in indirizzi fisici reali.
- L'applicazione deve poter garantire l'utilizzo di un *linguaggio espressivo* per formulare query atte alla personalizzazione dei parametri di visualizzazione delle entità.

1.1 Scenari

Allo stato attuale un utente medio potrebbe utilizzare il sistema tramite compilazione di un file di configurazione. La traduzione del suddetto file in un'interfaccia utente di configurazione più intuitiva è sicuramente nella visione globale dell'intero progetto, ma lasciata a sviluppi futuri. Infatti il nucleo pulsante di questo lavoro è considerato, da parte mia, la dinamicità e la libertà d'uso, ovvero il fatto di dare la possibilità all'utente di scegliere in qualunque momento *cosa* visualizzare e *come* visualizzarlo, dando a quest'ultima opzione il massimo della libertà possibile.

Presumendo che un utente compili il file di configurazione, esso potrebbe, per esempio, scegliere di visualizzare la regione geografica che desidera (e.g. Pechino, Cina), le entità che desidera monitorare (e.g. Veicoli), gli indici di selezione/evidenziazione di queste entità (e.g. Rumore, Vibrazioni, Gasolio, Ergonomia) e i loro criteri (e.g. terzili, quartili, o espressioni del tipo

$$\text{gasolio} \leq 30\%$$

). Potrebbe inoltre indicare quali caratteristiche delle entità visualizzare nella tabella dell'interfaccia (e.g. Tipo, Modello, Casa Manifatturiera, Targa, ecc.).

2 Analisi dei Requisiti

Alcuni requisiti non funzionali sono i seguenti:

- La comunicazione via rete tramite i componenti del sistema deve essere tale da non generare tempi di latenza eccessivi.
- Come tecnologia si vuole utilizzare uno stack MEVN (MongoDB, Express, Vue.js, Node.js), ma a causa dell'assenza di dati reali, si è evitato di instaurare un database fisico.

- Le entità e le loro caratteristiche sono simulate di volta in volta a run-time, in quanto non è necessario l'utilizzo di dati persistenti.

Il gap d'astrazione tra le tecnologie scelte e i problemi da risolvere non è eccessivo, in quanto sono state scelte appositamente le più recenti in ambito di programmazione di applicazioni web modulari. Un esempio lampante è la proprietà di *data-binding* di Vue.js che permette un'eccellente dinamicità e modularità nella costruzione dell'interfaccia utente e della generale programmazione lato client.

Il gap più elevato riscontrato è sicuramente quello relativo alle tecnologie di simulazione di dati realistici relativi a persone, veicoli, ecc.

3 Progettazione

L'architettura del sistema, come si nota in Figura 1, è composta da:

- *Frontend* realizzato con *Vue* come framework lato client e *Node.js*. Questo componente dell'architettura gestisce tutta l'interazione dell'utente con l'interfaccia (mappa, markers, tabella dei dati, ecc.), occupandosi di costruirla in base alle scelte dell'utilizzatore editate sul file di configurazione. Si occupa inoltre di comunicare, tramite chiamate HTTP REST, con il *backend* al fine di ottenere i dati richiesti dall'utente. L'ultima sua funzione è quella di instaurare un canale di comunicazione con il HERE-Development Server per ottenere un servizio di *reverse-geocoding* gratuito.

- *Backend* realizzato con *Express* e *Node.js*. Questo componente è il punto cardine della comunicazione tra tutti gli elementi del sistema. Esso infatti riceve i dati simulati in tempo reale dalle entità e le richieste da parte del frontend. Si occupa inoltre di gestire la maggior parte del carico di calcolo necessario per il corretto funzionamento delle funzionalità del sistema, come il processamento del linguaggio espressivo e il calcolo delle condizioni di selezione/monitoraggio espresse dinamicamente dall'utente.

Il motivo per il quale è stato scelto questo componente come gestore della maggior parte dell'attività di calcolo è stato il fatto di volere mantenere un client il più leggero possibile, oltre alla presa in considerazione delle possibilità di scalabilità di un componente software come il backend, soprattutto, anche se questo non è il caso, se si effettua la scelta di operare in termini di cloud computing.

- *Entità* realizzate con *Express* e *Node.js*. Questi componenti sono elementi indipendenti, il cui unico scopo è quello di simulare le attività/spostamenti delle entità con l'utilizzo di librerie come *Faker.js* e tracce gps pre-generate, e di inviare i dati al componente *backend*.
- *Servizio Reverse-Geocoding* è un servizio offerto da *HERE Development*, completamente gratuito. Il reverse-geocoding consiste nel tradurre coordinate geografiche in indirizzi fisici reali. La comunicazione tra il frontend e questo servizio avviene tramite chiamate HTTP REST.

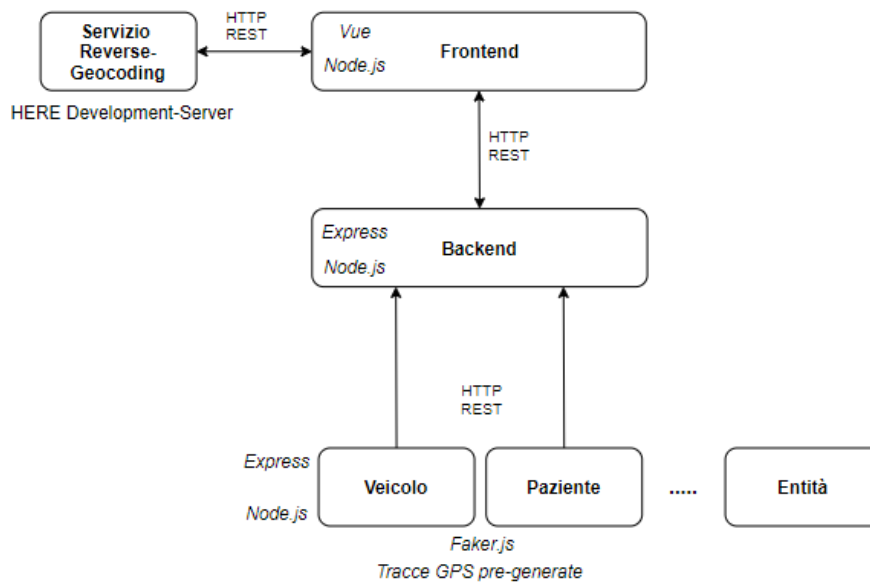


Figure 1: Architettura

3.1 Struttura

3.1.1 Frontend

Come si può evincere osservando la Figura 2, il frontend è costituito da diversi componenti *.vue*, fattore che permette di costruire l'interfaccia utente e tutte le sue funzionalità in maniera totalmente modulare, grazie alle proprietà gerarchiche dei componenti e della proprietà di *data-binding* del framework stesso. Quest'ultimo consente la comunicazione interna tra i componenti grazie ad un *event bus*, che propaga eventi scatenati dai componenti a tutti gli altri. Ogni evento generato può essere raccolto e gestito da tutti gli altri componenti, indipendentemente dal fatto che qualcuno di essi lo abbia fatto prima. Vi è quindi un'ottima gestione del bus degli eventi a basso livello, permettendo di non preoccuparsene, salvo rare eccezioni.

I dati prelevati dalla compilazione del file di configurazione da parte dell'utente vengono utilizzati da vari componenti:

- *App.vue* li utilizza per istanziare l'ambiente globale dell'applicazione, verificando la tipologia di entità che l'utente ha selezionato.
- *Page.vue* li utilizza per istanziare dinamicamente le quantità e le tipologie di componenti figli per costruire l'interfaccia utente.
- *Map.vue* li utilizza per istanziare dinamicamente le quantità e le tipologie di componenti figli per costruire la mappa dell'interfaccia utente e per determinare la logica dei dati che invia/riceve al/dal server.

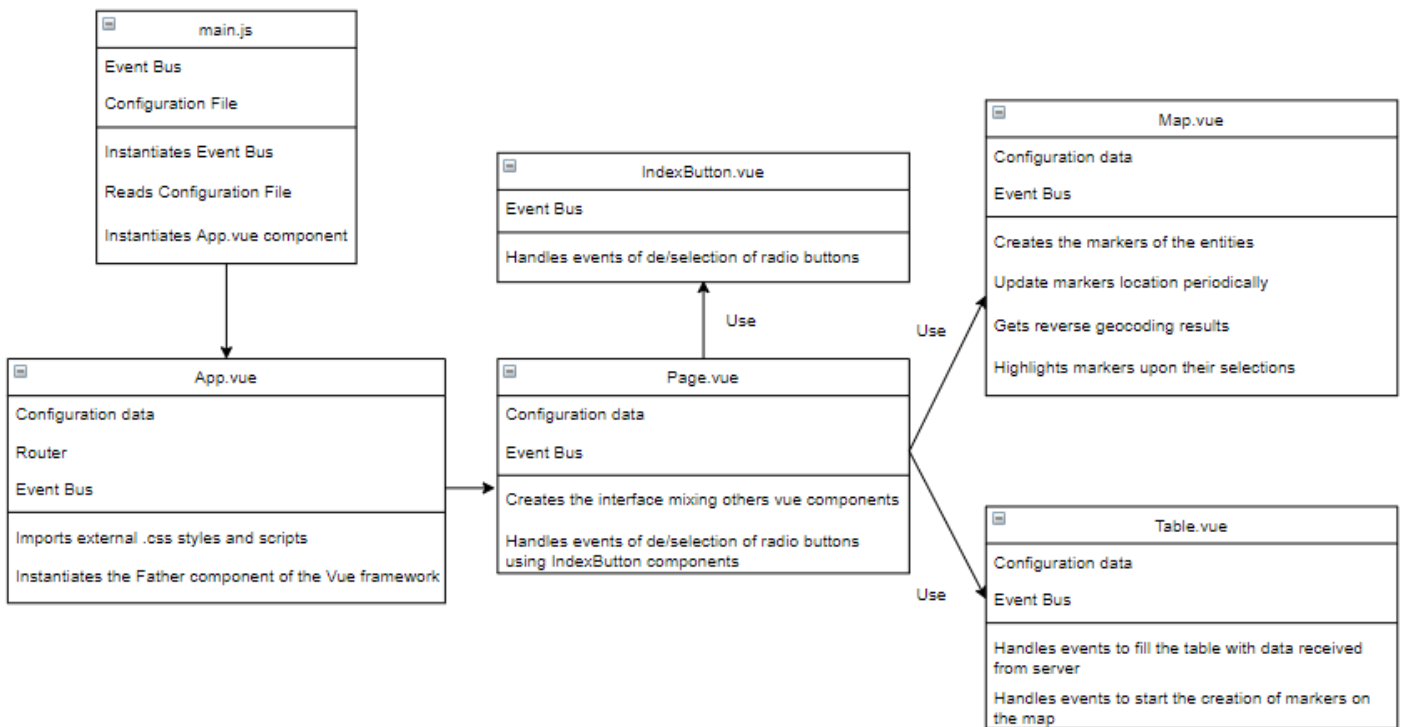


Figure 2: Frontend modellato

- *Table.vue* li utilizza per la costruzione della tabella dei dati da mostrare nell'interfaccia utente.

3.1.2 Backend

Come si può notare osservando la Figura 3, il backend è costituito da un file *app.js* che si occupa di configurare la porta per la comunicazione HTTP REST e di istanziare il file *service.js*. In questo file sono presenti e configurate tutte le API che gli altri componenti dell'architettura chiamano per comunicare con il backend e sfruttarne le funzionalità implementate.

La scelta di far ricadere su questo componente del sistema la maggior parte delle funzionalità di calcolo è dovuta al fatto, oltre al volere mantenere un frontend il più leggero possibile, che è possibile effettuare dei potenziamenti sia in *scale up* sia in *scale out*.

Ma la possibilità più vantaggiosa da prendere in considerazione, per quanto riguarda la scalabilità del sistema, è quella di implementare un supporto in termini di *cloud computing*. Questo permetterebbe un potenziale di scalabilità virtualmente infinito, oltre che ad alleggerire la gestione delle risorse di sistema. Esistono diversi tipi di servizi di cloud computing già avviati da sfruttare, come *Google Cloud Platform* o *Amazon AWS*, che

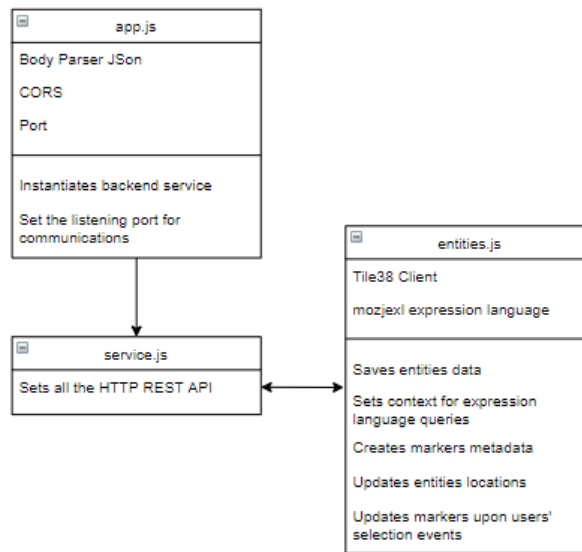


Figure 3: Backend modellato

permetterebbero così:

- Uno sviluppo più rapido, evitando vincoli o problemi relativi alla gestione sistemica.
- Di avere un protocollo di sicurezza dei dati già implementato.
- Di avere un possibile e più rapido accesso ad ulteriori funzionalità già presenti nel loro sistema.

Questo è sicuramente un punto da prendere in considerazione per quanto riguarda gli sviluppi futuri del progetto.

3.1.3 Entità

L'entità è l'elemento centrale della struttura del progetto, in quanto tutto è costruito per poter visualizzare e monitorare le sue caratteristiche e comportamenti a schermo. Come si può vedere in Figura 4, ogni entità dispone di un identificativo (Id), di un punto di partenza (Location), e di una serie di coordinate pre-generate e realistiche (Coordinates). Queste coordinate sono state estratte da un database Microsoft contenente una serie di tracce gps registrate da numerosi utenti di Pechino, Cina che hanno partecipato ad una iniziativa di ricerca.

Le entità specializzate sono i Veicoli e i Pazienti.

I dati di queste entità sono interamente generati automaticamente e pseudo-randomicamente a run-time, utilizzando la libreria *Faker.js*.

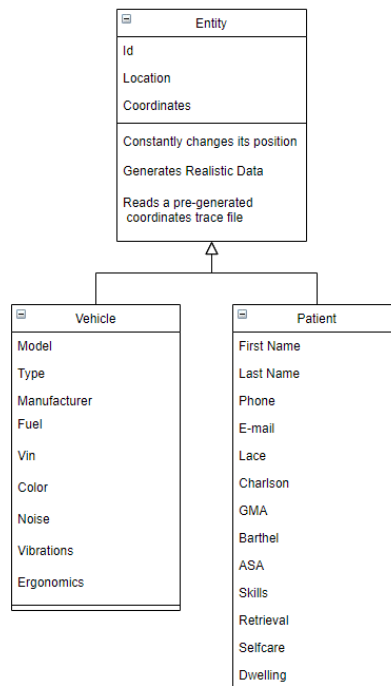


Figure 4: Entità modellate

3.1.4 Servizio Reverse-Geocoding

Il servizio scelto per implementare il reverse-geocoding è *HERE Developer*. HERE è un'impresa in proprietà delle aziende automobilistiche tedesche Audi, BMW e Daimler e fornisce servizi e tecnologie di dati di tipo geografico e di mappatura per il settore automobilistico, consumer e aziendale.

Vi sono numerosi enti che offrono questo tipo di servizio, ma la scelta è ricaduta su questo in quanto completamente gratuito e privo di limitazioni d'uso, offrendo una gamma di REST API in grado di soddisfare le esigenze di questo progetto.

La sua funzione, all'interno del sistema, è quella di ricevere richieste di reverse-geocoding (traduzione di coordinate geografiche in indirizzi fisici reali) e di inviare i risultati al sistema, che si occuperà di tenere aggiornate in tempo reale le informazioni dei markers delle entità con gli indirizzi fisici nei pressi dei quali esse si trovano.

3.2 Interazione

Come interagiscono, nel dettaglio, i componenti del sistema? La tecnologia di comunicazione principale è HTTP REST.

Come si può osservare dalla Figura 5 l'*utente* compila il file di configurazione, i cui dati vengono elaborati dal *frontend* per la costruzione dell'interfaccia utente e inviati al *back-end*, il quale risponderà con i markers da mostrare a schermo, completi con i metadati

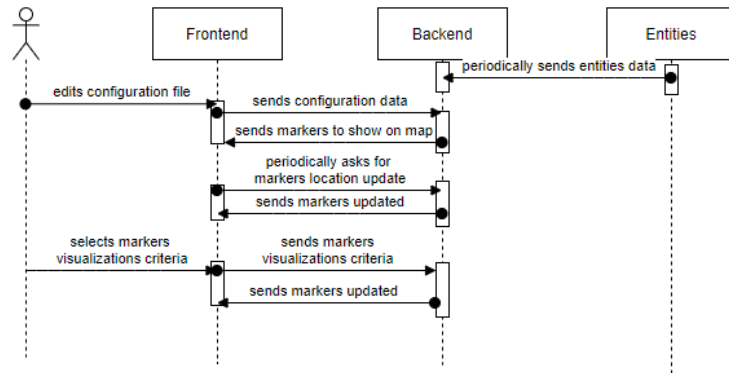


Figure 5: Interazione Utente-Frontend-Backend-Entità

delle entità scelte dall'utente.

La posizione geografica dei markers viene aggiornata periodicamente dal *backend* a fronte degli aggiornamenti che le *entità* simulate gli inviano, mentre lato *frontend* avviene un polling per richiedere questi dati in modo da aggiornare la posizione dei markers sulla mappa. Questa scelta è stata fatta a fronte del basso carico dati che ogni richiesta esige, in quanto si tratta di semplici coordinate geografiche. La scelta alternativa sarebbe stata quella di utilizzare le web sockets, ma a meno di far sì che il backend istanziasse un server figlio per la gestione della comunicazione, sarebbero state bloccanti. Considerato il fatto che il backend è il nucleo centrale di calcolo del sistema e data la bassa mole di dati coinvolta, inserire una meccanica bloccante non è sembrata la scelta più efficiente.

Test sull'efficienza della struttura di comunicazione sono stati effettuati e i suoi risultati verranno mostrati alla fine di questa sottosezione.

Come si osserva dalla Figura 6, l'interazione con il *Server HERE-Developer* per il reverse-geocoding avviene periodicamente a partire dal *frontend*, il quale, a seguito dell'aggiornamento delle coordinate dei markers, invia una richiesta al servizio per tradurre le nuove coordinate in indirizzi fisici reali.

3.2.1 Test di efficienza

I test effettuati sono stati incrementalmente lineari, partendo da un carico di venti entità coinvolte e finendo con un centinaio di elementi.

Come si può notare in Figura 7 al crescere lineare delle entità si è verificato un aumento lineare dei tempi di latenza della comunicazione, che è quello che ci si aspettava. Inoltre, i tempi risultanti sono considerati più che accettabili dal punto di vista dell'efficienza.

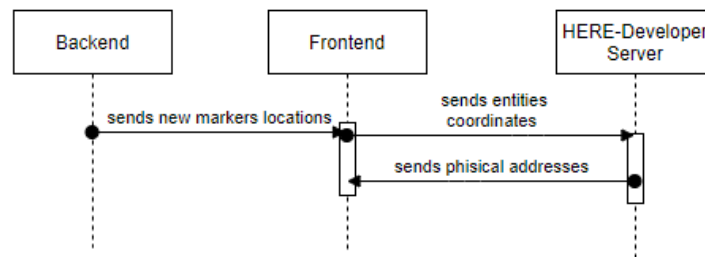


Figure 6: Interazione Frontend-HEREServer

20 Entità	10-12 millisecondi	
40 Entità	20-24 millisecondi	
60 Entità	44-64 millisecondi	
80 Entità	62-80 millisecondi	
100 Entità	81-96 millisecondi	

Figure 7: Tests sull'efficienza

4 Analisi sulla Distribuzione

Allo stato attuale, il sistema è stato utilizzato e testato sulla stessa macchina, effettuando delle chiamate HTTP REST in *localhost*. Se si volesse spostare, per esempio, il *backend* su una macchina differente basterebbe cambiare l'indirizzo fisico codificato in un file di configurazione.

Per quanto riguarda la possibilità di avere più utenti connessi, al momento è possibile avere più utenti connessi sulla stessa macchina, ma con una limitazione. Il backend risponde alle chiamate multiple (nel caso avessi più frontend aperti) alla stessa macchina e quindi, gli eventuali risultati chiesti da un client vengono visualizzati anche sull'altro. Questo risultato è dovuto dalla mancanza di un protocollo di identificazione (di utente e/o di macchina) che rende unica una sessione d'uso.

Un'osservazione può essere fatta anche dal punto di vista della possibilità di avere più istanze del backend su macchine diverse. Allo stato attuale è possibile e come già affermato precedentemente l'unico vincolo risiede nell'indirizzo fisico da indicare al frontend, in quanto al momento non è presente un protocollo di scelta dinamica di un'istanza del backend più appropriata, per esempio dal punto di vista geografico per la minimizzazione della latenza.

Si può inoltre precisare che sarebbe possibile disporre di istanze del backend relative ad aree geografiche differenti, in quanto il componente, allo stato attuale non ha alcun vincolo geografico, dovuto al fatto dell'assenza di un controllo dinamico sulla localizzazione geografica delle entità. Tuttavia, non disponendo di un database fisico associato al backend, e non essendoci un'associazione con l'area geografica scelta dinamicamente dall'utente nel frontend, non risulta prettamente fattibile allo stato attuale.

5 Dettagli di Implementazione

In questa sezione si vuole mostrare lo stato attuale del file di configurazione, ovvero un semplice file *.json*, contenente, come mostrato in Figura 8:

- *center*: il punto geografico in cui la mappa deve essere centrata.
- *entity*: la tipologia di entità che si sceglie di visualizzare.
- *indexes*: gli indici per i quali l'utente può cambiare i criteri di visualizzazione dei markers sulla mappa.
- *values*: i valori/caratteristiche che l'utente vuole vedere nella tabella dei dati su schermo.

Si vuole mostrare, inoltre, la potenza e la semplicità implementativa della proprietà di *data-binding* del framework *Vue*. Come mostrato in Figura 9, in base al numero di elementi scelti dall'utente in fase di editing del file di configurazione, vengono istanziati

```

"center": [39.984702,116.318417],
"entity": "vehicles",
"indexes": [
  { "component": "IndexButton", "text": "Noise", "criterion": "tertile" },
  { "component": "IndexButton", "text": "Vibrations", "criterion": "quartile" },
  { "component": "IndexButton", "text": "Fuel", "criterion": "vehicles[.fuel < 30]" },
  { "component": "IndexButton", "text": "Ergonomics", "criterion": "vehicles[.ergonomics < 30]" }
],
"values": [
  { "text": "Vehicle", "value": "vehicle" },
  { "text": "Manufacturer", "value": "manufacturer" },
  { "text": "Model", "value": "model" },
  { "text": "Type", "value": "type" },
  { "text": "Vin", "value": "vin" },
  { "text": "Noise (db/A)", "value": "noise" },
  { "text": "Vibrations (m/s^2)", "value": "vibrations" },
  { "text": "Fuel %", "value": "fuel" },
  { "text": "Ergonomics %", "value": "ergonomics" }
],

```

Figure 8: File di Configurazione

```

<div>
  <template v-for="block in myjson_indexes">
    <component :is="block.component" :block="block" :key="block.text"></component>
  </template>
</div>

```

Figure 9: Esempio di Data-Binding di Vue

un numero corrispettivo di componenti minori che andranno a costituire l'interfaccia personalizzata e dinamica.

6 Esempi d'Uso

Prendendo come esempio d'uso il file di configurazione mostrato in figura 8, il risultato a schermo è quello in Figura 10 e in Figura 11 (Il contenuto della tabella dei valori è zoomato in Figura 12 e in Figura 13).

Al centro della schermata è visibile la mappa centrata alle coordinate indicate (Pechino, Cina) e i markers dei cento veicoli simulati. Sulla parte sinistra vi sono gli indici selezionabili. Sotto la mappa è osservabile la tabella con i valori delle caratteristiche delle entità.

6.1 Indicatori nel dettaglio

Come si osserva dalla Figura 14, selezionando l'indice *Noise*, il sistema, basandosi sui valori simulati associati a ciascuna entità, colora i markers basandosi su criteri diversi,

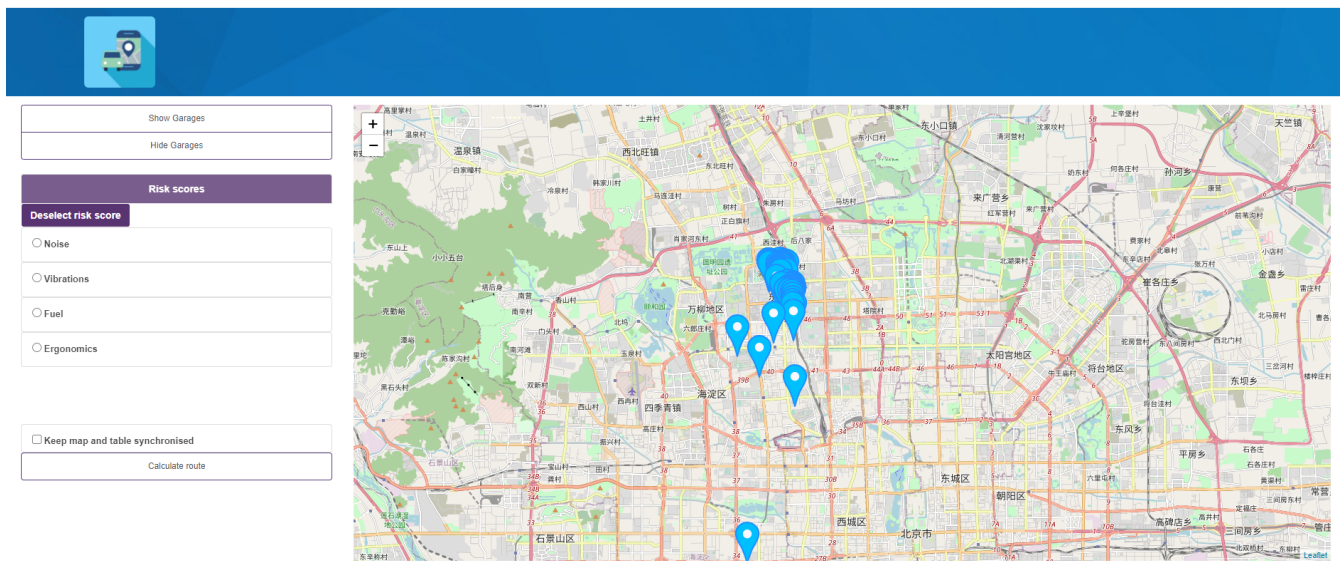


Figure 10: Schermata risultante dopo la configurazione

☐	Vehicle	Manufacturer	Model	Type	Vin	Noise (dba)	Vibrations (m/s ²)	Fuel %	Ergonomics %
☐	Mini Altima	Ferrari	LeBaron	Coupe	JPPHFSZEOKM543290	75.99	0.48	82.30	97.20
☐	Fiat Land Cruiser	Nissan	CTS	Extended Cab Pickup	2PELQMF83NF57865	79.80	0.33	99.83	81.86
☐	Mazda El Camino	Smart	Grand Caravan	Passenger Van	LCQFBNDD88HZ86788	13.70	0.44	56.10	81.32
☐	Bentley Jetta	Hyundai	Focus	Extended Cab Pickup	H5F2BWFVWOY155185	25.85	0.54	27.88	89.78
☐	Maserati Model S	BMW	El Camino	Crew Cab Pickup	EOVXG3HHZ8NF85181	71.48	0.46	82.03	82.84

Rows per page: 5 1-5 of 100

Figure 11: Tabella dei valori

Search (UPPER CASE ONLY)

☐	Vehicle	Manufacturer	Model	Type	Vin
☐	Mini Altima	Ferrari	LeBaron	Coupe	JPPHFSZEOKM545290
☐	Fiat Land Cruiser	Nissan	CTS	Extended Cab Pickup	2PELQMF83NF57865
☐	Mazda El Camino	Smart	Grand Caravan	Passenger Van	LCQFBNDD88HZ86788
☐	Bentley Jetta	Hyundai	Focus	Extended Cab Pickup	H5F2BWFVWOY155185
☐	Maserati Model S	BMW	El Camino	Crew Cab Pickup	EOVXG3HHZ8NF85181

Figure 12: Tabella dei valori - Zoom prima parte

Noise (db/A)	Vibrations (m/s^2)	Fuel %	Ergonomics %
75.95	0.46	62.30	37.20
79.60	0.33	98.83	61.89
13.70	0.44	59.10	61.32
25.65	0.54	27.68	89.78
73.48	0.48	92.03	52.84

Rows per page: 5 1-5 of 100

Figure 13: Tabella dei valori - Zoom seconda parte

che possono essere quartili, terzili o arbitrari. In questo caso, come da file di configurazione, è stato scelto un criterio a terzili, quindi vengono colorati in *verde* i veicoli meno rumorosi, in *giallo* quelli con emissioni di rumore medie, ed in *rosso* quelli più rumorosi.

6.1.1 Linguaggio espressivo

Come mostrato in Figura 15, selezionando l'indice *Fuel*, al quale è associato un criterio costruito con il linguaggio espressivo *vehicles[.fuel ≤ 30]*, i veicoli colorati sono solamente quelli i cui valori associati di *fuel* sono effettivamente minori del 30%.

6.2 Cambio di entità - da Veicoli a Pazienti

Nel momento nel quale l'utente decide di volere visualizzare entità diverse, per esempio i pazienti di un ospedale, il risultato a schermo, dopo la configurazione, sarebbe quello mostrato in Figura 16 e in Figura 17, con le stesse funzionalità presentate nella sezione precedente in ambito dei veicoli.

6.3 Reverse geo-coding nel dettaglio

Come si può vedere dalla Figura 18 il risultato del servizio di *reverse geo-coding* si nota cliccando su un marker, facendo apparire un popup dove vengono indicate alcune informazioni dell'entità e il suo indirizzo reale nel quale sta transitando.

7 Conclusioni

Concludendo, il lavoro svolto è stato sicuramente soddisfacente per quanto riguarda il raggiungimento degli obiettivi proposti. Chiaramente vi sono ancora numerose migliorie

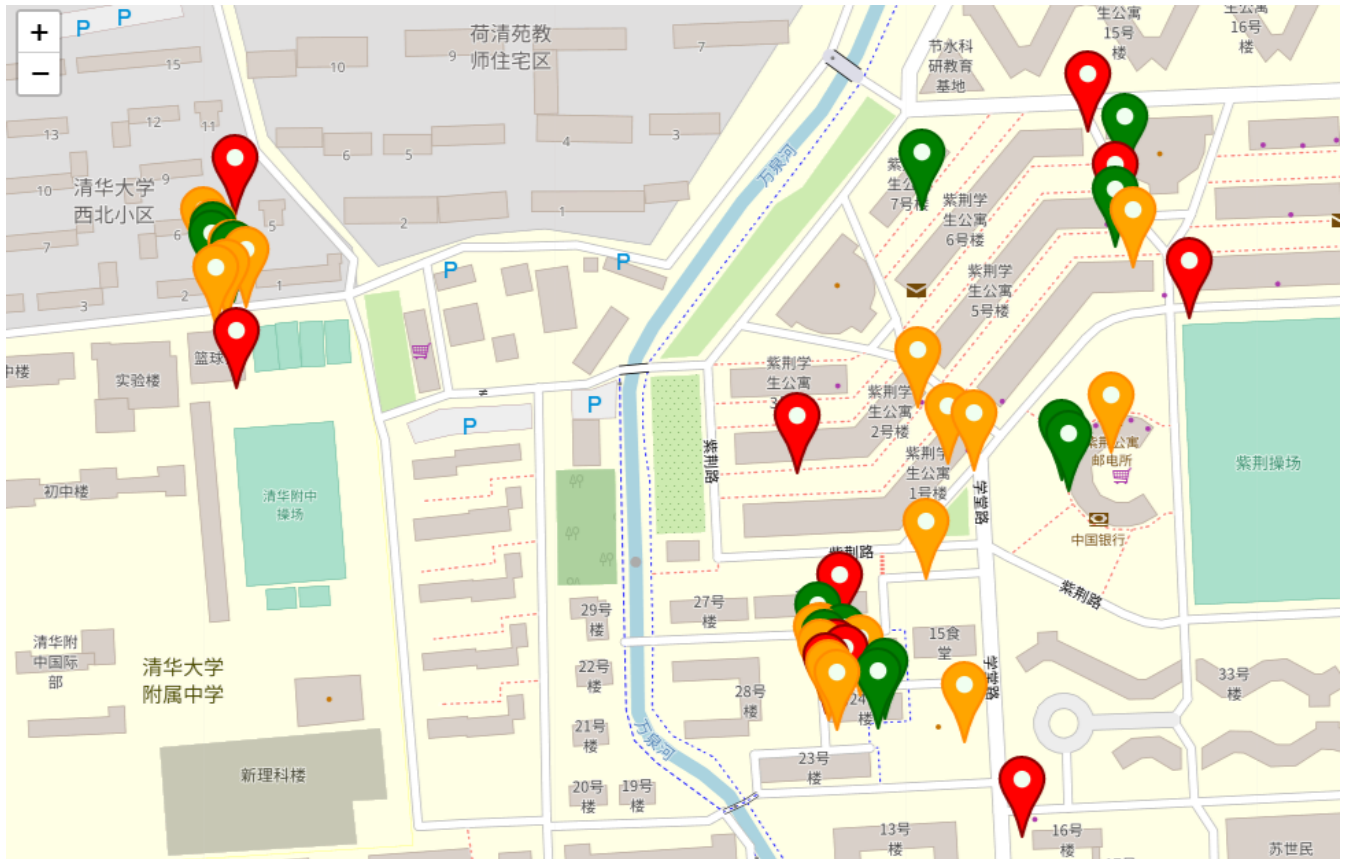


Figure 14: Indicatori nel dettaglio - Noise



Figure 15: Indicatori nel dettaglio - Linguaggio espressivo con Fuel

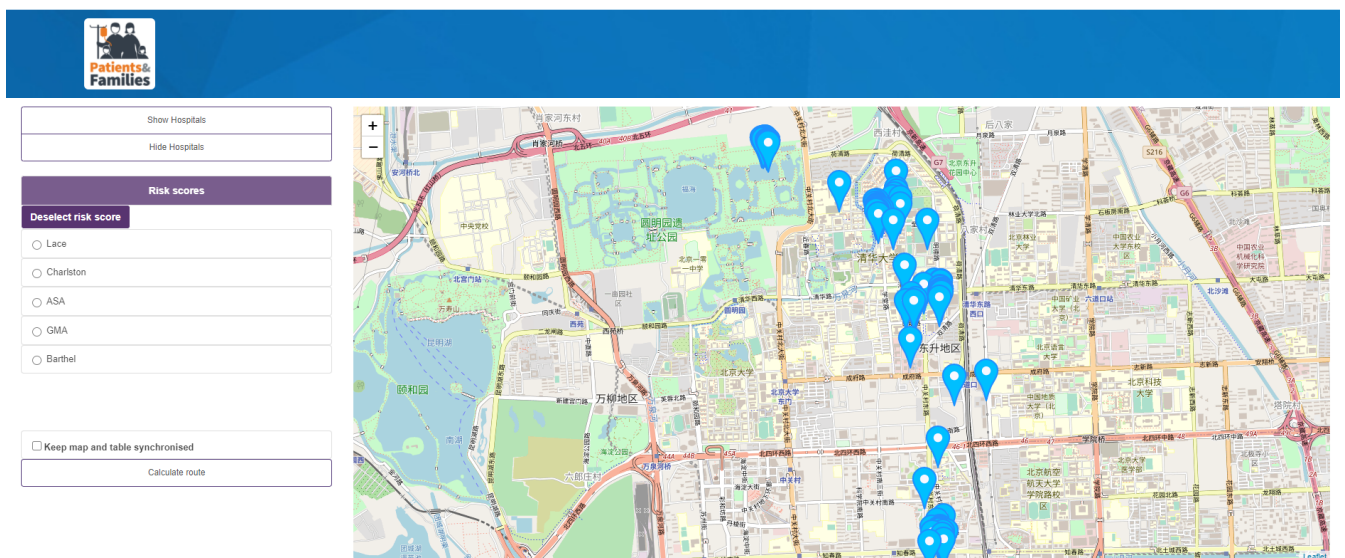


Figure 16: Schermata Pazienti

☐	Name	Surname	Retrieval	Selfcare	Dwelling	Career	LACE	Charleston	Berthel	ASA	GMA
☐	吉	東部	YES	0	2	0	74	4.84	34	II	4
☐	保太	中村	YES	1	2	1	79	2.11	94	II	0
☐	世	鈴木	YES	0	3	1	75	1.43	32	I	1
☐	裕次	中村	NO	2	3	1	10	3.73	88	II	1
☐	耀司	鈴木	YES	0	3	1	28	1.88	73	III	2

Rows per page: 5 1-5 of 100 < >

Figure 17: Tabella Pazienti

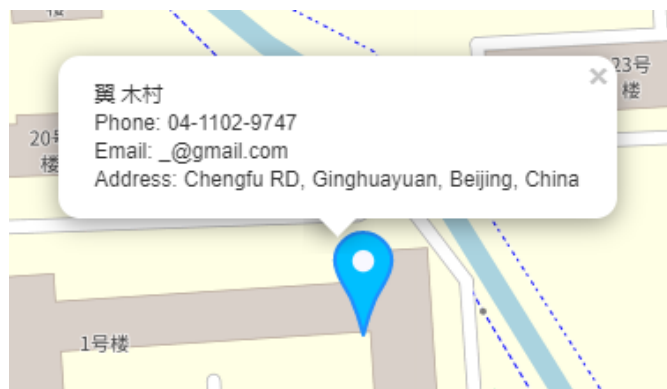


Figure 18: Reverse geo-coding

da poter effettuare e funzionalità da implementare. Esse sono indicate nella prossima sezione.

7.1 Lavori Futuri

Tra i lavori futuri si possono inserire i seguenti:

- Costruzione di un'interfaccia utente di configurazione che nasconda la complessità ora richiesta, nella compilazione del file, con semplici menu interattivi;
- Esplorazione delle tecniche di geo-fencing, ovvero triggers geo-localizzati configurabili per il monitoraggio del comportamento delle entità;
- Ampliamento delle tipologie di entità gestite dal sistema;
- Ritrovamento o generazione di dati reali da inserire in un database fisico per completare lo stack MEVN.
- A fronte del punto di cui sopra, rimuovere la componente simulata, avvicinandosi ad un ambiente reale.
- Considerare l'opzione di spostarsi verso un'implementazione *cloud computing*